Relación de Ejercicios de Clases y Objetos (9): Sockets y Threads

En esta relación, vamos a conectar unos programas con otros por medio de Sockets, que es un elemento básico para conexiones por red. Más adelante, veremos como ejecutar varios hilos dentro del mismo programa.

1) Servidor y Cliente de Bonoloto

En este primer ejercicio de la relación, tendremos que crear un par de proyectos. El primero será un servidor que realizará sorteos de la bonoloto y nos enviará los seis números premiados. El segundo será un cliente que se conectará al servidor y recibirá los seis números.

El servidor tendrá la siguiente clase:

- **ServidorBonoloto**
  - Atributo estático:
    - *Random* r, para evitar que salgan siempre los mismos números.
  - Método estático:
    - lanzaServidor() lanzará el servidor. El funcionamiento será muy simple:
      - Abriremos un *ServerSocket* en el puerto 9009 para que se conecten nuestros clientes y entraremos en un while.
      - Cuando se conecte un cliente, sortearemos los 6 números (metiéndolos primero en una lista, como siempre) y los escribiremos por el *Socket*.

El cliente tendrá la siguiente clase:

- **ClienteBonoloto**
  - Atributo:
    - *int*[] boleto, en donde guardaremos los seis números que escribamos en nuestro boleto de la bonoloto.
  - Constructor:
    - Tendrá un constructor al que le pasaremos un array con los 6 números que hayamos elegido para nuestro boleto. Habrá que comprobar que el array tenga 6 enteros y que estén todos entre el 1 y el 49.
  - Método:
    - *int* conectaCliente(*String* ip) se conectará al servidor y recibirá los seis enteros que nos mandará que será el resultado del sorteo.
      A continuación, comprobará cuántos de esos números tenemos en nuestro boleto y nos devolverá el número de aciertos.

2) Servidor y Cliente de Mayúsculas

En el segundo ejercicio, escribiremos un servidor al que se podrán conectar los clientes para escribir texto y el servidor les devolverá el mismo texto en mayúsculas. El servidor deberá usar threads (hilos) para que se puedan conectar varios clientes al mismo tiempo. También los crearemos en proyectos separados.

El servidor tendrá la siguiente clase:

- **ServidorMayusculas**
    - Método estático:
        - lanzaServidor() lanzará el Servidor. En este caso, será un servidor multihilo, así que deberá crear una *ThreadPool* y ejecutar una instancia de la clase interna. El *ServerSocket* lo abrirá en el puerto 9010.
    - Clase interna: **HiloServidor** que implementa un *Runnable*
        - Atributo: *Socket* socket
        - Constructor al que le pasamos un Socket y lo guarda en el atributo.
        - Método: run() que es el que gestionará la conexión del cliente. Leerá las cadenas que le mande el cliente, las pondrá en mayúsculas y se las enviará de vuelta al cliente. Acabará y cerrará el *Socket* cuando reciba una cadena vacía.

El cliente tendrá la siguiente clase:

- **ClienteMayusculas**
    - Método estático:
        - conectaCliente(*String* ip) se conectará al servidor en la IP que le pasamos por parámetro. Nos irá pidiendo cadenas que enviará al servidor. Después de enviarlas, leerá la línea que le devuelve el servidor. Terminará cuando escribamos una cadena vacía.

3) Programa de chat entre dos clientes

En este tercer ejercicio escribiremos un cliente que se podrá conectar a otro y enviar y recibir mensajes de chat. El cliente usará un hilo para poder recibir los mensajes mientras está enviando los suyos al otro cliente.

- **ChatDirecto**
  - Métodos estáticos:
    - escuchar(): abrirá un *ServerSocket* en el puerto 9011 y esperará a que se conecte un cliente. Después lanzará un *Thread* de *HiloRecibir* para leer los mensajes. Por último, pedirá al usuario líneas y las irá escribiendo por un *PrintWriter* hasta que el usuario escriba una línea vacía, momento en el que parará el *HiloRecibir* con un *stop* y cerrará el *Socket*.
    - conectar(*String* ip): muy parecido al escuchar salvo que en lugar de esperar a que se conecten, se conectará a otro que esté escuchando.
  - Clase interna: **HiloRecibir** que implementa un *Runnable*
    - Atributos:
      - *Socket* socket
      - *volatile boolean* salir, que nos servirá para indicarle al hilo cuándo tiene que cerrarse. Valdrá *false* de inicio.
    - Constructor al que le pasamos un Socket y lo guarda en el atributo.
    - Métodos:
      - run(), que se queda en un bucle while mientras *salir* sea *false* leyendo líneas que llegan por el socket y poniéndolas por pantalla.
      - stop(), que pone la variable *salir* a *true* para que termine la ejecución del método *run*.

4) Servidor de chat en grupo

En el cuarto y último ejercicio de esta relación, escribiremos un servidor de chat al que se pueden conectar varios clientes simultáneamente. Estos clientes podrán enviar mensajes al servidor y el servidor los reenviará al resto de clientes, de manera similar a cómo funciona un grupo de WhatsApp. Necesitaremos crear un par de proyectos: uno para el servidor y otro para el cliente.

- **ServidorChat**
  Para el servidor, necesitaremos crear un hilo para cada cliente que se nos conecte, pero, además, necesitaremos poder mandar mensajes al resto de clientes que estén conectados, así que necesitaremos guardar la información necesaria de cada cliente que nos permita poder mandar mensajes. La forma más simple es guardar una lista con el *PrintWriter* de cada cliente.

  - Atributos estáticos:
    - listaWriters: una lista donde guardaremos los *PrintWriter* de cada cliente.
  - Métodos estáticos:
    - lanzaServidor(): nos lanzará el servidor en el puerto 9012 y creará un hilo para cada cliente que se conecte.
  - Clase interna:
    - Clase interna: **HiloServidor** que implementa un *Runnable*
      - Atributo: *Socket* socket
      - Constructor al que le pasamos un Socket y lo guarda en el atributo.
      - Método: run() que es el que gestionará la conexión de cada cliente. Al inicio, deberá leer la primera cadena que le manda el cliente, que será el nombre de usuario; deberá mandar un mensaje al grupo de que el cliente se ha conectado indicando su nombre. Después, cada línea que reciba del cliente la reenviará al grupo hasta que reciba una línea con el comando "/salir", momento en el que mandará una notificación de que el usuario se ha desconectado y acabará el proceso.

- **ClienteChat**
  - Métodos estáticos:
    - conectar(*String* ip, *String* usuario): Se conectará a la IP que le pasemos en el puerto 9012. Después deberá lanzar un hilo independiente para poder recibir los mensajes, usando la clase HiloRecibir. A continuación, enviará al servidor el nombre de usuario que vamos a usar y entrará en un bucle en el que irá leyendo líneas por consola y enviándolas al servidor. Saldremos del bucle al escribir el comando "/salir", momento en el que se cerrará la conexión.

- Clase interna: **HiloRecibir** que implementa un *Runnable*
  - Atributos:
    - *Socket* socket
    - *volatile boolean* salir, que nos servirá para indicarle al hilo cuándo tiene que cerrarse. Valdrá *false* de inicio.
  - Constructor al que le pasamos un Socket y lo guarda en el atributo.
  - Métodos:
    - run(), que se queda en un bucle while mientras *salir* sea *false* leyendo líneas que llegan por el socket y poniéndolas por pantalla.
    - stop(), que pone la variable *salir* a *true* para que termine la ejecución del método *run*.